Study guide: Nonlinear differential equation problems

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Nov 10, 2014

What makes a differential equations nonlinear?

- In linear differential equations, the unknown u or its derivatives appear in linear terms $au(t)$, $au'(t)$, $a\nabla^2 u$, where a is independent of u.
- All other types of terms containing u are *nonlinear* and contain products of u or its derivatives.

examples on linear and nonlinear differential equations

Linear ODE:

$$u'(t) = a(t)u(t) + b(t)$$

Nonlinear ODE:

$$u'(t) = u(t)(1 - u(t)) = u(t) - u(t)^2$$

This (pendulum) ODE is also nonlinear:

$$u'' + \gamma \sin u = 0$$

because

$$\sin u = u - \frac{1}{6}u^3 + \mathcal{O}(u^5),$$

contains products of u

Introduction of basic concepts

- Logistic ODE as simple model for a nonlinear problem
- Introduction of basic techniques:
  - Explicit time integration (no nonlinearities)
  - Implicit time integration (nonlinearities)
  - Linearization and Picard iteration
  - Linearization via Newton's method
  - Linearization via a trick like geometric mean
- Numerical illustration of the performance

The scaled logistic ODE

$$u'(t) = u(t)(1-u(t)) = u - u^2$$

Linearization by explicit time discretization

Forward Euler method:

$$\frac{u^{n+1}-u^n}{\Delta t} = u^n(1-u^n)$$

gives a *linear* algebraic equation for the unknown value u^{n+1}!

Explicit time integration methods will (normally) linearize a nonlinear problem.
Another example: 2nd-order Runge-Kutta method

$$\begin{aligned} u^* &= u^n + \Delta t u^n(1-u^n), \\ u^{n+1} &= u^n + \Delta t \frac{1}{2}\left(u^n(1-u^n) + u^*(1-u^*)\right) . \end{aligned}$$

An implicit method: Backward Euler discretization

A backward time difference

$$\frac{u^n-u^{n-1}}{\Delta t} = u^n(1-u^n)$$

gives a *nonlinear* algebraic equation for the unknown u^n. The equation is of quadratic type (which can easily be solved exactly):

$$\Delta t(u^n)^2 + (1-\Delta t)u^n - u^{n-1} = 0$$

Detour: new notation

To make formulas less overloaded and the mathematics as close as possible to computer code, a new notation is introduced:

- $u^{(1)}$ means u^{n-1}
- In general: $u^{(\ell)}$ means $u^{n-\ell}$
- u is the unknown (u^n)

Nonlinear equation to solve in new notation:

$$F(u) = \Delta t u^2 + (1 - \Delta t)u - u^{(1)} = 0$$

Exact solution of quadratic nonlinear equations

Solution of $F(u) = 0$:

$$u = \frac{1}{2\Delta t}\left(-1 - \Delta t \pm \sqrt{(1 - \Delta t)^2 - 4\Delta t u^{(1)}}\right)$$

Observation: Nonlinear algebraic equations may have multiple solutions!

How do we pick the right solution in this case?

Let's investigate the nature of the two roots:

```
>>> import sympy as sp
>>> dt, u_1, u = sp.symbols('dt u_1 u')
>>> r1, r2 = sp.solve(dt*u**2 + (1-dt)*u - u_1, u)  # find roots
>>> r1
(dt - sqrt(dt**2 + 4*dt*u_1 - 2*dt + 1) - 1)/(2*dt)
>>> r2
(dt + sqrt(dt**2 + 4*dt*u_1 - 2*dt + 1) - 1)/(2*dt)
>>> print r1.series(dt, 0, 2)
-1/dt + 1 - u_1 + dt*(u_1**2 - u_1) + O(dt**2)
>>> print r2.series(dt, 0, 2)
u_1 + dt*(-u_1**2 + u_1) + O(dt**2)
```

The `r1` root behaves as $1/\Delta t \to \infty$ as $\Delta t \to 0$! Therefore, only the `r2` root is of relevance.

Linearization

- In general, we cannot solve nonlinear algebraic equations with formulas
- We must *linearize* the equation, or create a recursive set of *linearized* equations whose solutions hopefully converge to the solution of the nonlinear equation
- Manual linearization may be an art

- Automatic linearization is possible (cf. Newton's method)

Examples will illustrate the points!

Picard iteration

Nonliner equation from Backward Euler scheme for logistic ODE:

$$F(u) = au^2 + bu + c = 0$$

Let u^- be an available approximation of the unknown u.
Linearization of u^2: $u^- u$

$$F(u) \approx \hat{F}(u) = au^- u + bu + c = 0$$

But

- Problem: the solution u of $\hat{F}(u) = 0$ is not the exact solution of $F(u) = 0$
- Solution: set $u^- = u$ and repeat the procedure

The algorithm of Picard iteration

At a time level, set $u^- = u^{(1)}$ (solution at previous time level) and iterate:

$$u = -\frac{c}{au^- + b}, \quad u^- \leftarrow u$$

This technique is known as

- fixed-point iteration
- successive substitutions
- nonlinear Richardson iteration
- Picard iteration

The algorithm of Picard iteration with classical math notation

- u^k: computed approximation in iteration k
- u^{k+1} is the next approximation (unknown)

$$au^k u^{k+1} + bu^{k+1} + c = 0 \quad \Rightarrow \quad u^{k+1} = -\frac{c}{au^k + b}, \quad k = 0, 1, \ldots$$

Or with a time level n too:

$$au^{n,k} u^{n,k+1} + bu^{n,k+1} - u^{n-1} = 0 \quad \Rightarrow \quad u^{n,k+1} = \frac{u^n}{au^{n,k} + b}, \quad k = 0, 1, \ldots$$

Stopping criteria

Using change in solution:

$$|u - u^-| \le \epsilon_u$$

or change in residual:

$$|F(u)| = |au^2 + bu + c| < \epsilon_r$$

A single Picard iteration

Common simple and cheap technique: perform 1 single Picard iteration

$$\frac{u^n - u^{n-1}}{\Delta t} = u^n(1 - u^{n-1})$$

Inconsistent time discretization ($u(1-u)$ must be evaluated for n, $n-1$, or $n-\frac{1}{2}$) - can produce quite inaccurate results, but is very popular.

Implicit Crank-Nicolson discretization

Crank-Nicolson discretization:

$$[D_t u = u(1-u)]^{n+\frac{1}{2}}$$

$$\frac{u^{n+1} - u^n}{\Delta t} = u^{n+\frac{1}{2}} - (u^{n+\frac{1}{2}})^2$$

Approximate $u^{n+\frac{1}{2}}$ as usual by an arithmetic mean,

$$u^{n+\frac{1}{2}} \approx \frac{1}{2}(u^n + u^{n+1})$$

$$(u^{n+\frac{1}{2}})^2 \approx \frac{1}{4}(u^n + u^{n+1})^2 \quad \text{(nonlinear term)}$$

which is nonlinear in the unknown u^{n+1}.

Linearization by a geometric mean

Using a *geometric mean* for $(u^{n+\frac{1}{2}})^2$ linearizes the nonlinear term $(u^{n+\frac{1}{2}})^2$ (error $\mathcal{O}(\Delta t^2)$ as in the discretization of u'):

$$(u^{n+\frac{1}{2}})^2 \approx u^n u^{n+1}$$

Arithmetic mean on the linear $u^{n+\frac{1}{2}}$ term and a geometric mean for $(u^{n+\frac{1}{2}})^2$ gives a linear equation for u^{n+1}:

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}(u^n + u^{n+1}) + u^n u^{n+1}$$

Note: Here we turned a nonlinear algebraic equation into a linear one. No need for iteration! (Consistent $\mathcal{O}(\Delta t^2)$ approx.)

Newton's method

Write the nonlinear algebraic equation as

$$F(u) = 0$$

Newton's method: linearize $F(u)$ by two terms from the Taylor series,

$$\begin{aligned} F(u) &= F(u^-) + F'(u^-)(u - u^-) + \frac{1}{2}F''(u^-)(u - u^-)^2 + \cdots \\ &\approx F(u^-) + F'(u^-)(u - u^-) = \hat{F}(u) \end{aligned}$$

The linear equation $\hat{F}(u) = 0$ has the solution

$$u = u^- - \frac{F(u^-)}{F'(u^-)}$$

Newton's method with an iteration index

$$u^{k+1} = u^k - \frac{F(u^k)}{F'(u^k)}, \quad k = 0, 1, \ldots$$

Newton's method exhibits *quadratic convergence* if u^k is sufficiently close to the solution. Otherwise, the method may diverge.

Using Newton's method on the logistic ODE

$$F(u) = au^2 + bu + c$$

$$F'(u) = 2au + b$$

The iteration method becomes

$$u = u^- + \frac{a(u^-)^2 + bu^- + c}{2au^- + b}, \quad u^- \leftarrow u$$

Start of iteration: $u^- = u^{(1)}$

Using Newton's method on the logistic ODE with typical math notation

Set iteration start as $u^{n,0} = u^{n-1}$ and iterate with explicit indices for time (n) and Newton iteration (k):

$$u^{n,k+1} = u^{n,k} + \frac{\Delta t(u^{n,k})^2 + (1-\Delta t)u^{n,k} - u^{n-1}}{2\Delta t u^{n,k} + 1 - \Delta t}$$

Compare notation with

$$u = u^{-} + \frac{\Delta t(u^{-})^2 + (1-\Delta t)u^{-} - u^{(1)}}{2\Delta t u^{-} + 1 - \Delta t}$$

Relaxation may improve the convergence

- Problem: Picard and Newton iteration may change the solution too much
- Remedy: relaxation (less change in the solution)
- Let u^* be the suggested new value from Picard or Newton iteration

Relaxation with *relaxation parameter* ω (weight old and new value):

$$u = \omega u^* + (1-\omega)u^{-}, \quad \omega \le 1$$

Simple formula when used in Newton's method:

$$u = u^{-} - \omega\frac{F(u^{-})}{F'(u^{-})}$$

Implementation; part 1

Program `logistic.py`

```
def BE_logistic(u0, dt, Nt, choice='Picard',
                eps_r=1E-3, omega=1, max_iter=1000):
    if choice == 'Picard1':
        choice = 'Picard';  max_iter = 1

    u = np.zeros(Nt+1)
    iterations = []
    u[0] = u0
    for n in range(1, Nt+1):
        a = dt
        b = 1 - dt
        c = -u[n-1]

        if choice == 'Picard':

            def F(u):
                return a*u**2 + b*u + c

            u_ = u[n-1]
```

```
            k = 0
            while abs(F(u_)) > eps_r and k < max_iter:
                u_ = omega*(-c/(a*u_ + b)) + (1-omega)*u_
                k += 1
            u[n] = u_
            iterations.append(k)
```

Implementation; part 2

```
def BE_logistic(u0, dt, Nt, choice='Picard',
                eps_r=1E-3, omega=1, max_iter=1000):
    ...
        elif choice == 'Newton':

            def F(u):
                return a*u**2 + b*u + c

            def dF(u):
                return 2*a*u + b

            u_ = u[n-1]
            k = 0
            while abs(F(u_)) > eps_r and k < max_iter:
                u_ = u_ - F(u_)/dF(u_)
                k += 1
            u[n] = u_
            iterations.append(k)
    return u, iterations
```

Implementation; part 3

The Crank-Nicolson method with a geometric mean:

```
def CN_logistic(u0, dt, Nt):
    u = np.zeros(Nt+1)
    u[0] = u0
    for n in range(0, Nt):
        u[n+1] = (1 + 0.5*dt)/(1 + dt*u[n] - 0.5*dt)*u[n]
    return u
```

Experiments: accuracy of iteration methods

Experiments: number of iterations

The effect of relaxation can potentially be great!

- $\Delta t = 0.9$: Picard required 32 iterations on average
- $\omega = 0.8$: 7 iterations
- $\omega = 0.5$: 2 iterations (!) - optimal choice

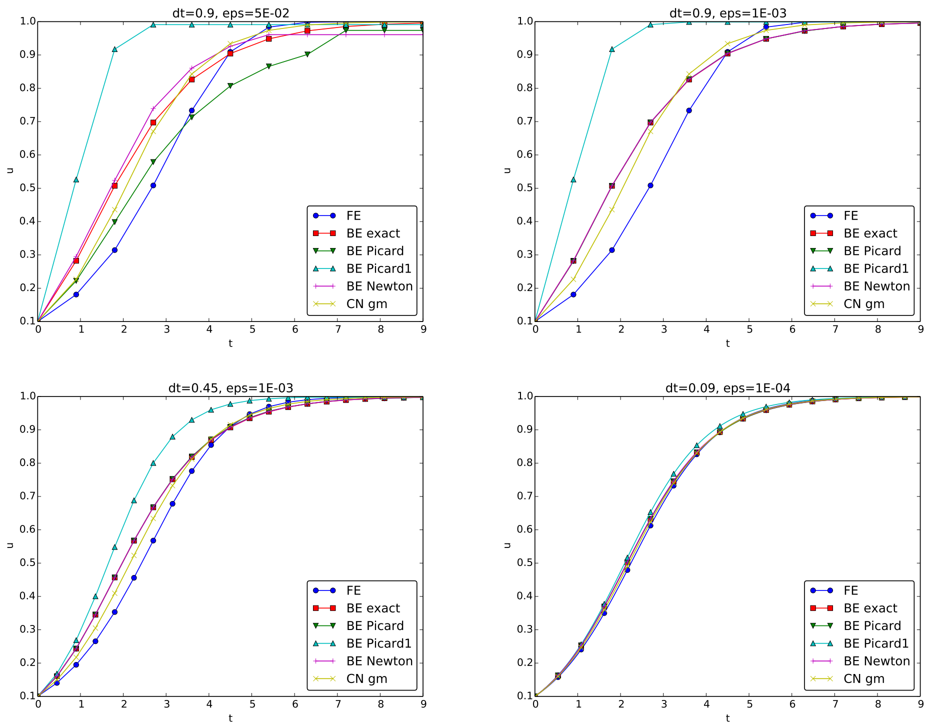

Figure 1: The impact of solution strategies and for four different time step lengths on the solution.

Other $\omega = 1$ experiments:

Δt	ϵ_r	Picard	Newton
0.2	10^{-7}	5	2
0.2	10^{-3}	2	1
0.4	10^{-7}	12	3
0.4	10^{-3}	4	2
0.8	10^{-7}	58	3
0.8	10^{-3}	4	2

Generalization to a general nonlinear ODE

$$u' = f(u, t)$$

Note: f is in general nonlinear in u so the ODE is nonlinear

Explicit time discretization

Forward Euler and all explicit methods sample f with known values and all nonlinearities are gone:

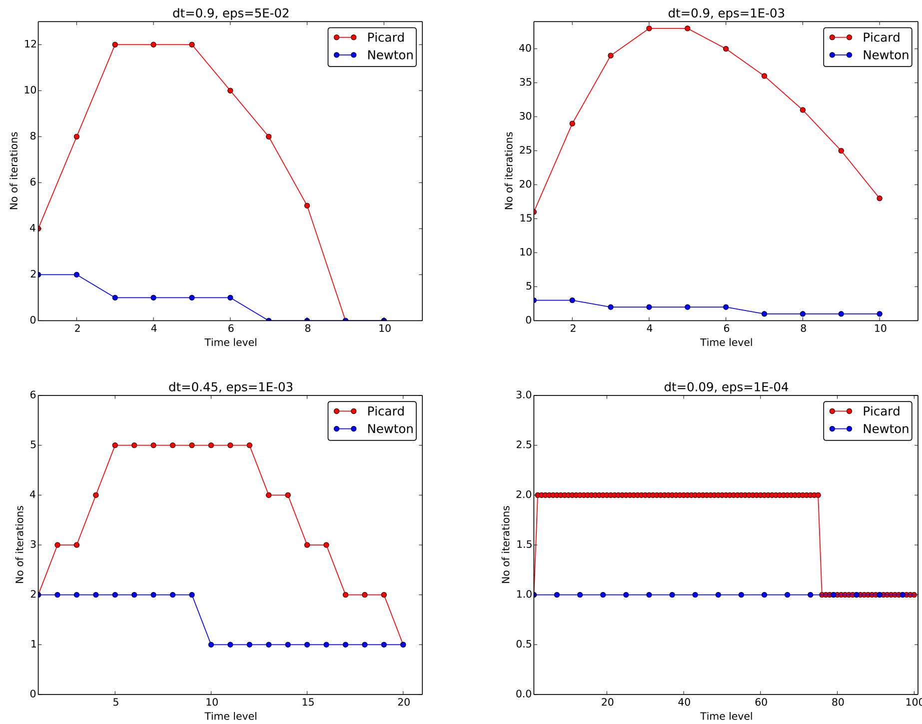

Figure 2: Comparison of the number of iterations at various time levels for Picard and Newton iteration.

$$\frac{u^{n+1} - u^n}{\Delta t} = f(u^n, t_n)$$

Backward Euler discretization

Backward Euler $[D_t^- u = f]^n$ leads to nonlinear algebraic equations:

$$F(u^n) = u^n - \Delta t\, f(u^n, t_n) - u^{n-1} = 0$$

Alternative notation:

$$F(u) = u - \Delta t\, f(u, t_n) - u^{(1)} = 0$$

Picard iteration for Backward Euler scheme

A simple Picard iteration, not knowing anything about the nonlinear structure of f, must approximate $f(u, t_n)$ by $f(u^-, t_n)$:

$$\hat{F}(u) = u - \Delta t\, f(u^-, t_n) - u^{(1)}$$

The iteration starts with $u^- = u^{(1)}$ and proceeds with repeating

$$u^* = \Delta t\, f(u^-, t_n) + u^{(1)}, \quad u = \omega u^* + (1-\omega)u^-, \quad u^- \;\leftarrow\; u$$

until a stopping criterion is fulfilled.

Manual linearization for a given $f(u,t)$

- $f(u^-, t)$: *explicit* treatment of f
  (as in time-discretization)
- $f(u,t)$: *fully implicit* treatment of f
- If f has some structure, say $f(u,t) = u^3$, we may think of a *partially implicit* treatment: $(u^-)^2 u$
- More implicit treatment of f often gives faster convergence
  (as it gives more stable time discretizations)

Trick for partially implicit treatment of a general $f(u,t)$:

$$f(u^-, t)\frac{u}{u^{-1}}$$

(Idea: $u \approx u^-$)

Computational experiments with partially implicit treatment of f

- $f(u,t) = -u^3$:
  - $(u^-)^3$ linearization: 22, 9, 6 iterations
  - $(u^-)^2 u$ linearization: 8, 5, 4 iterations
- $f(u,t) = e^{-u}$: a trick $f(u^-, t)u/u^-$ has no effect
- $f(u,t) = \sin(2(u+1))$: a trick $f(u^-, t)u/u^-$ has effect
  (7, 9, 11 iterations vs 17, 21, 20)

Newton's method for Backward Euler scheme

Newton's method requires the computation of the derivative

$$F'(u) = 1 - \Delta t \frac{\partial f}{\partial u}(u, t_n)$$

Algorithm for Newton's method for $u' = f(u,t)$. Start with $u^- = u^{(1)}$, then iterate

$$u = u^- - \omega\frac{F(u^-)}{F'(u^-)} = u^- - \omega\frac{u^{(1)} + \Delta t\, f(u^-, t_n)}{1 - \Delta t\frac{\partial}{\partial u}f(u^-, t_n)}$$

Crank-Nicolson discretization

The standard Crank-Nicolson scheme with arithmetic mean approximation of f reads

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}(f(u^{n+1}, t_{n+1}) + f(u^n, t_n))$$

Nonlinear algebraic equation:

$$F(u) = u - u^{(1)} - \Delta t\frac{1}{2}f(u, t_{n+1}) - \Delta t\frac{1}{2}f(u^{(1)}, t_n) = 0$$

Picard and Newton iteration in the Crank-Nicolson case

Picard iteration (for a general f):

$$\hat{F}(u) = u - u^{(1)} - \Delta t\frac{1}{2}f(u^-, t_{n+1}) - \Delta t\frac{1}{2}f(u^{(1)}, t_n)$$

Newton's method:

$$F(u) = u - u^{(1)} - \Delta t\frac{1}{2}f(u, t_{n+1}) - \Delta t\frac{1}{2}f(u^{(1)}, t_n)$$

$$F'(u) = 1 - \frac{1}{2}\Delta t\frac{\partial f}{\partial u}(u, t_{n+1})$$

Systems of ODEs

$$\begin{aligned}
\frac{d}{dt}u_0(t) &= f_0(u_0(t), u_1(t), \ldots, u_N(t), t)\\
\frac{d}{dt}u_1(t) &= f_1(u_0(t), u_1(t), \ldots, u_N(t), t),\\
&\vdots\\
\frac{d}{dt}u_N(t) &= f_N(u_0(t), u_1(t), \ldots, u_N(t), t)
\end{aligned}$$

Introduce vector notation:

- $u = (u_0(t), u_1(t), \ldots, u_N(t))$
- $(f_0(u,t), f_1(u,t), \ldots, f_N(u,t))$

Vector form:

$$u' = f(u,t), \quad u(0) = U_0$$

Schemes: apply scalar scheme to each component

A Backward Euler scheme for the vector ODE $u' = f(u,t)$

$$\begin{aligned}
\frac{u_0^n - u_0^{n-1}}{\Delta t} &= f_0(u^n, t_n)\\
\frac{u_1^n - u_1^{n-1}}{\Delta t} &= f_1(u^n, t_n)\\
&\vdots\\
\frac{u_N^n - u_N^{n-1}}{\Delta t} &= f_N(u^n, t_n)
\end{aligned}$$

This can be written more compactly in vector form as

$$\frac{u^n - u^{n-1}}{\Delta t} = f(u^n, t_n)$$

This is a system of *nonlinear algebraic equations*,

$$u^n - \Delta t\, f(u^n, t_n) - u^{n-1} = 0,$$

or written out

$$\begin{aligned}
u_0^n - \Delta t\, f_0(u^n, t_n) - u_0^{n-1} &= 0,\\
&\vdots\\
u_N^n - \Delta t\, f_N(u^n, t_n) - u_N^{n-1} &= 0\,.
\end{aligned}$$

Example: Crank-Nicolson scheme for the oscillating pendulum model

The scaled equations for an oscillating pendulum:

$$\dot{\omega} = -\sin\theta - \beta\omega|\omega|, \tag{1}$$

$$\dot{\theta} = omega, \tag{2}$$

Set $u_0 = \omega$, $u_1 = \theta$

$$\begin{aligned}
u_0' &= f_0(u,t) = -\sin u_1 - \beta u_0|u_0|,\\
u_1' &= f_1(u,t) = u_1\,.
\end{aligned}$$

Crank-Nicolson discretization:

$$\frac{u_0^{n+1} - u_0^n}{\Delta t} = -\sin u_1^{n+\frac{1}{2}} - \beta u_0^{n+\frac{1}{2}}|u_0^{n+\frac{1}{2}}| \approx -\sin\left(\frac{1}{2}(u_1^{n+1} + u_1 n)\right) - \beta\frac{1}{4}(u_0^{n+1} + u_0^n)|u_0^{n+1} + u_0^n|, \tag{3}$$

$$\frac{u_1^{n+1} - u_1^n}{\Delta t} = v_0^{n+\frac{1}{2}} \approx \frac{1}{2}(u_0^{n+1} + u_0^n)\,. \tag{4}$$

The nonlinear 2×2 system

Introduce u_0 and u_1 for u_0^{n+1} and u_1^{n+1}, write $u_0^{(1)}$ and $u_1^{(1)}$ for u_0^n and u_1^n, and rearrange:

$$F_0(u_0, u_1) = u_0 - u_0^{(1)} + \Delta t\,\sin\left(\frac{1}{2}(u_1 + u_1^{(1)})\right) + \frac{1}{4}\Delta t\beta(u_0 + u_0^{(1)})|u_0 + u_0^{(1)}| = 0$$

$$F_1(u_0, u_1) = u_1 - u_1^{(1)} - \frac{1}{2}\Delta t(u_0 + u_0^{(1)}) = 0$$

Systems of nonlinear algebraic equations

$$\boxed{\begin{aligned} x\cos y + y^3 &= 0 \\ y^2 e^x + xy &= 2 \end{aligned}}$$

Systems of nonlinear algebraic equations arise from solving *systems of ODEs* or solving *PDEs*

Notation for general systems of algebraic equations

$$F(u) = 0$$

where

$$u = (u_0, \ldots, u_N), \quad F = (F_0, \ldots, F_N)$$

Special linear system-type structure
(arises frequently in PDE problems):

$$A(u)u = b(u)$$

Picard iteration

Picard iteration for $F(u)=0$ is meaningless unless there is some structure so we can linearize. For $A(u)u=b(u)$ we can linearize

$$A(u^-)u = b(u^-)$$

Note: we solve a system of nonlinear algebraic equations as a sequence of linear systems.

Algorithm for relaxed Picard iteration

Given $A(u)u=b(u)$ and an initial guess u^-, iterate until convergence:

1. solve $A(u^-)u^* = b(u^-)$ with respect to u^*
2. $u = \omega u^* + (1-\omega)u^-$
3. $u^- \ \leftarrow \ u$

"Until convergence": $||u-u^-|| \le \epsilon_u$ or $||A(u)u-b|| \le \epsilon_r$

Newton's method for $F(u)=0$

Linearization of $F(u)=0$ equation via multi-dimensional Taylor series:

$$F(u) = F(u^-) + J(u^-)\cdot(u-u^-) + \mathcal{O}(||u-u^-||^2)$$

where J is the *Jacobian* of F, sometimes denoted $\nabla_u F$, defined by

$$J_{i,j} = \frac{\partial F_i}{\partial u_j}$$

Approximate the original nonlinear system $F(u)=0$ by

$$\hat{F}(u) = F(u^-) + J(u^-)\cdot\delta u = 0, \quad \delta u = u-u^-$$

which is linear vector equation in u

Algorithm for Newton's method

$$\underline{F(u^-)}_{\text{vector}} + \underline{J(u^-)}_{\text{matrix}} \cdot \underline{\delta u}_{\text{vector}} = 0$$

Solution by a two-step procedure:

1. solve linear system $J(u^-)\delta u = -F(u^-)$ wrt δu
2. update $u = u^- + \delta u$

Relaxed update:

$$u = \omega(u^- + \delta u) + (1-\omega)u^- = u^- + \omega\delta u$$

Newton's method for $A(u)u = b(u)$

For

$$F_i = \sum_k A_{i,k}(u)u_k - b_i(u)$$

one gets

$$J_{i,j} = \frac{\partial F_i}{\partial u_j} = \sum_k \frac{\partial A_{i,k}}{\partial u_j} u_k + A_{i,j} - \frac{\partial b_i}{\partial u_j}$$

Matrix form:

$$(A + A'u + b')\delta u = -Au + b$$

$$(A(u^-) + A'(u^-)u^- + b'(u^-))\delta u = -A(u^-)u^- + b(u^-)$$

Comparison of Newton and Picard iteration

Newton:

$$(A(u^-) + A'(u^-)u^- + b'(u^-))\delta u = -A(u^-)u^- + b(u^-)$$

Rewrite:

$$\underbrace{A(u^-)(u^- + \delta u) - b(u^-)}_{\text{Picard system}} + \gamma(A'(u^-)u^- + b'(u^-))\delta u = 0$$

All the "Picard terms" are contained in the Newton formulation.

Combined Picard-Newton algorithm

Idea: Write a common Picard-Newton algorithm so we can trivially switch between the two methods (e.g., start with Picard, get faster convergence with Newton when u is closer to the solution)

Algorithm: Given $A(u)$, $b(u)$, and an initial guess u^-, iterate until convergence:

1. solve $(A + \gamma(A'(u^-)u^- + b'(u^-)))\delta u = -A(u^-)u^- + b(u^-)$ with respect to δu
2. $u = u^- + \omega \delta u$
3. $u^- \leftarrow u$

Note:

- $\gamma = 1$: Newton's method
- $\gamma = 0$: Picard iteration

Stopping criteria

Let $||\cdot||$ be the standard Eucledian vector norm. Several termination criteria are much in use:

- Absolute change in solution: $||u - u^-|| \le \epsilon_u$
- Relative change in solution: $||u - u^-|| \le \epsilon_u ||u_0||$, where u_0 denotes the start value of u^- in the iteration
- Absolute residual: $||F(u)|| \le \epsilon_r$
- Relative residual: $||F(u)|| \le \epsilon_r ||F(u_0)||$
- Max no of iterations: stop when $k > k_{\max}$

Combination of absolute and relative stopping criteria

Problem with relative criterion: a small $||F(u_0)||$ (because $u_0 \approx u$, perhaps because of small Δt) must be significantly reduced. Better with absolute criterion.

- Can make combined absolute-relative criterion
- ϵ_{rr}: tolerance for relative part
- ϵ_{ra}: tolerance for absolute part

$$||F(u)|| \le \epsilon_{rr} ||F(u_0)|| + \epsilon_{ra}$$

$$||F(u)|| \le \epsilon_{rr} ||F(u_0)|| + \epsilon_{ra} \quad \text{or} \quad ||\delta u|| \le \epsilon_{ur} ||u_0|| + \epsilon_{ua} \quad \text{or} \quad k > k_{\max}$$

Example: A nonlinear ODE model from epidemiology

Spreading of a disease (e.g., a flu) can be modeled by a 2×2 ODE system

$$\begin{aligned} S' &= -\beta S I \\ I' &= \beta S I - \nu I \end{aligned}$$

Here:

- $S(t)$ is the number of people who can get ill (susceptibles)
- $I(t)$ is the number of people who are ill (infected)
- Must know $\beta > 0$ (danger of getting ill) and $\nu > 0$ ($1/\nu$: expected recovery time)

Implicit time discretization

A Crank-Nicolson scheme:

$$\frac{S^{n+1}-S^n}{\Delta t} = -\beta[SI]^{n+\frac{1}{2}} \approx -\frac{\beta}{2}(S^nI^n + S^{n+1}I^{n+1})$$
$$\frac{I^{n+1}-I^n}{\Delta t} = \beta[SI]^{n+\frac{1}{2}} - \nu I^{n+\frac{1}{2}} \approx \frac{\beta}{2}(S^nI^n + S^{n+1}I^{n+1}) - \frac{\nu}{2}(I^n + I^{n+1})$$

New notation: S for S^{n+1}, $S^{(1)}$ for S^n, I for I^{n+1}, $I^{(1)}$ for I^n

$$F_S(S,I) = S - S^{(1)} + \frac{1}{2}\Delta t\beta(S^{(1)}I^{(1)} + SI) = 0$$
$$F_I(S,I) = I - I^{(1)} - \frac{1}{2}\Delta t\beta(S^{(1)}I^{(1)} + SI) - \frac{1}{2}\Delta t\nu(I^{(1)} + I) = 0$$

A Picard iteration

- We have approximations S^- and I^- to S and I.
- Linearize SI in S ODE as I^-S (linear equation in S!)
- Linearize SI in I ODE as S^-I (linear equation in I!)

$$S = \frac{S^{(1)} - \frac{1}{2}\Delta t\beta S^{(1)}I^{(1)}}{1 + \frac{1}{2}\Delta t\beta I^-}$$
$$I = \frac{I^{(1)} + \frac{1}{2}\Delta t\beta S^{(1)}I^{(1)}}{1 - \frac{1}{2}\Delta t\beta S^- + \nu}$$

Before a new iteration: $S^- \leftarrow S$ and $I^- \leftarrow I$

Newton's method

$$F(u) = 0, \quad F = (F_S, F_I),\ u = (S, I)$$

Jacobian:

$$J = \begin{pmatrix} \frac{\partial}{\partial S}F_S & \frac{\partial}{\partial I}F_S \\ \frac{\partial}{\partial S}F_I & \frac{\partial}{\partial I}F_I \end{pmatrix} = \begin{pmatrix} 1 + \frac{1}{2}\Delta t\beta I & \frac{1}{2}\Delta t\beta \\ -\frac{1}{2}\Delta t\beta S & 1 - \frac{1}{2}\Delta t\beta I - \frac{1}{2}\Delta t\nu \end{pmatrix}$$

Newton system: $J(u^-)\delta u = -F(u^-)$

$$\begin{pmatrix} 1 + \frac{1}{2}\Delta t\beta I^- & \frac{1}{2}\Delta t\beta S^- \\ -\frac{1}{2}\Delta t\beta S^- & 1 - \frac{1}{2}\Delta t\beta I^- - \frac{1}{2}\Delta t\nu \end{pmatrix}\begin{pmatrix} \delta S \\ \delta I \end{pmatrix} =$$
$$\begin{pmatrix} S^- - S^{(1)} + \frac{1}{2}\Delta t\beta(S^{(1)}I^{(1)} + S^-I^-) \\ I^- - I^{(1)} - \frac{1}{2}\Delta t\beta(S^{(1)}I^{(1)} + S^-I^-) - \frac{1}{2}\Delta t\nu(I^{(1)} + I^-) \end{pmatrix}$$

Actually no need to bother with nonlinear algebraic equations for this particular model...

Remark: For this particular system of ODEs, explicit time integration methods work very well. Even a Forward Euler scheme is fine, but the 4-th order Runge-Kutta method is an excellent balance between high accuracy, high efficiency, and simplicity.

Linearization at the differential equation level

Goal: linearize a PDE like

$$\frac{\partial u}{\partial t} = \nabla\cdot(\alpha(u)\nabla u) + f(u)$$

PDE problem

$$\begin{aligned}
\frac{\partial u}{\partial t} &= \nabla\cdot(\alpha(u)\nabla u) + f(u), & \boldsymbol{x}\in\Omega,\ t\in(0,T] \\
-\alpha(u)\frac{\partial u}{\partial n} &= g, & \boldsymbol{x}\in\partial\Omega_N,\ t\in(0,T] \\
u &= u_0, & \boldsymbol{x}\in\partial\Omega_D,\ t\in(0,T]
\end{aligned}$$

Explicit time integration

$$\boxed{\text{Explicit time integration methods remove the nonlinearity}}$$

Forward Euler method:

$$[D_t^+ u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^n$$

$$\frac{u^{n+1}-u^n}{\Delta t} = \nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n)$$

This is a *linear equation* in the unknown $u^{n+1}(\boldsymbol{x})$, with solution

$$u^{n+1} = u^n + \Delta t\nabla\cdot(\alpha(u^n)\nabla u^n) + \Delta t f(u^n)$$

Disadvantage: $\Delta t \leq (\max\alpha)^{-1}(\Delta x^2+\Delta y^2+\Delta z^2)$

Backward Euler scheme

Backward Euler scheme:

$$[D_t^- u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^n$$

Written out:

$$\frac{u^n - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n)$$

This is a nonlinear, stationary PDE for the unknown function $u^n(\boldsymbol{x})$

Picard iteration for Backward Euler scheme

We have

$$\frac{u^n - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n)$$

Picard iteration:

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k})$$

Start iteration with $u^{n,0} = u^{n-1}$

Picard iteration with alternative notation

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k})$$

Rewrite with a simplified, implementation-friendly notation:

- u means the unknown $u^{n,k+1}$ to solve for
- u^- means the most recent approximation to u
- $u^{(1)}$ means u^{n-1} ($u^{(\ell)}$ means $u^{n-\ell}$)

$$\frac{u - u^{(1)}}{\Delta t} = \nabla\cdot(\alpha(u^-)\nabla u) + f(u^-)$$

Start iteration with $u^- = u^{(1)}$; update with u^- to u.

Backward Euler scheme and Newton's method

> Normally, Newton's method is defined for systems of *algebraic equations*, but the idea of the method can be applied at the PDE level too!

Let $u^{n,k}$ be an approximation to the unknown u^n. We seek a better approximation

$$u^n = u^{n,k} + \delta u$$

- Insert $u^n = u^{n,k} + \delta u$ in the PDE
- Taylor expand the nonlinearities and keep only terms that are linear in δu

Result: linear PDE for the *approximate* correction δu

Calculation details of Newton's method at the PDE level

Insert $u^{n,k} + \delta u$ for u^n in PDE:

$$\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k} + \delta u)\nabla(u^{n,k} + \delta u)) + f(u^{n,k} + \delta u)$$

Taylor expand $\alpha(u^{n,k} + \delta u)$ and $f(u^{n,k} + \delta u)$:

$$\begin{aligned}
\alpha(u^{n,k} + \delta u) &= \alpha(u^{n,k}) + \frac{d\alpha}{du}(u^{n,k})\delta u + \mathcal{O}(\delta u^2) \approx \alpha(u^{n,k}) + \alpha'(u^{n,k})\delta u \\
f(u^{n,k} + \delta u) &= f(u^{n,k}) + \frac{df}{du}(u^{n,k})\delta u + \mathcal{O}(\delta u^2) \approx f(u^{n,k}) + f'(u^{n,k})\delta u
\end{aligned}$$

Calculation details of Newton's method at the PDE level

Inserting linear approximations of α and f:

$$\begin{aligned}
\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} = {} & \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{n,k}) + \\
& \nabla \cdot (\alpha(u^{n,k})\nabla \delta u) + \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla u^{n,k}) + \\
& \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla \delta u) + f'(u^{n,k})\delta u
\end{aligned}$$

Note: $\alpha'(u^{n,k})\delta u \nabla \delta u$ is $\mathcal{O}(\delta u^2)$ and therefore omitted.

Result of Newton's method at the PDE level

$$\delta F(\delta u; u^{n,k}) = -F(u^{n,k})$$

with

$$\begin{aligned}
F(u^{n,k}) &= \frac{u^{n,k} - u^{n-1}}{\Delta t} - \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{n,k}) \\
\delta F(\delta u; u^{n,k}) &= -\frac{1}{\Delta t}\delta u + \nabla \cdot (\alpha(u^{n,k})\nabla \delta u) + \\
& \quad \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla u^{n,k}) + f'(u^{n,k})\delta u
\end{aligned}$$

Note:

- δF is linear in δu
- F contains only known terms

Similarity with Picard iteration

Rewrite the PDE for δu using $u^{n,k} + \delta u = u^{n,k+1}$:

$$\begin{aligned}\frac{u^{n,k+1} - u^{n-1}}{\Delta t} &= \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k}) \\ &+ \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla u^{n,k}) + f'(u^{n,k})\delta u\end{aligned}$$

Note:

- The first line is the same PDE as arise in the Picard iteration
- The remaining terms arise from the differentiations in Newton's method

Using new notation for implementation

- u for u^n
- u^- for $u^{n,k}$
- $u^{(1)}$ for u^{n-1}

$$\delta F(\delta u; u^-) = -F(u^-) \quad \text{(PDE)}$$

$$\begin{aligned}F(u^-) &= \frac{u^- - u^{(1)}}{\Delta t} - \nabla \cdot (\alpha(u^-)\nabla u^-) + f(u^-) \\ \delta F(\delta u; u^-) &= -\frac{1}{\Delta t}\delta u + \nabla \cdot (\alpha(u^-)\nabla \delta u) + \\ &\quad \nabla \cdot (\alpha'(u^-)\delta u \nabla u^-) + f'(u^-)\delta u\end{aligned}$$

Combined Picard and Newton formulation

$$\begin{aligned}\frac{u - u^{(1)}}{\Delta t} &= \nabla \cdot (\alpha(u^-)\nabla u) + f(u^-) + \\ &\quad \gamma(\nabla \cdot (\alpha'(u^-)(u - u^-)\nabla u^-) + f'(u^-)(u - u^-))\end{aligned}$$

Observe:

- $\gamma = 0$: Picard iteration
- $\gamma = 1$: Newton's method

Why? Easy to switch (start with Picard, use Newton close too solution)

Crank-Nicolson discretization

Crank-Nicolson discretization applies a centered difference at $t_{n+\frac{1}{2}}$:

$$[D_t u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^{n+\frac{1}{2}}\,.$$

Many choices of formulating an arithmetic means:

$$\begin{aligned}
[f(u)]^{n+\frac{1}{2}} &\approx f(\frac{1}{2}(u^n + u^{n+1})) = [f(\overline{u}^t)]^{n+\frac{1}{2}}\\
[f(u)]^{n+\frac{1}{2}} &\approx \frac{1}{2}(f(u^n) + f(u^{n+1})) = [\overline{f(u)}^t]^{n+\frac{1}{2}}\\
[\alpha(u)\nabla u]^{n+\frac{1}{2}} &\approx \alpha(\frac{1}{2}(u^n + u^{n+1}))\nabla(\frac{1}{2}(u^n + u^{n+1})) = \alpha(\overline{u}^t)\nabla\overline{u}^t]^{n+\frac{1}{2}}\\
[\alpha(u)\nabla u]^{n+\frac{1}{2}} &\approx \frac{1}{2}(\alpha(u^n) + \alpha(u^{n+1}))\nabla(\frac{1}{2}(u^n + u^{n+1})) = [\overline{\alpha(u)}^t\nabla\overline{u}^t]^{n+\frac{1}{2}}\\
[\alpha(u)\nabla u]^{n+\frac{1}{2}} &\approx \frac{1}{2}(\alpha(u^n)\nabla u^n + \alpha(u^{n+1})\nabla u^{n+1}) = [\overline{\alpha(u)\nabla u}^t]^{n+\frac{1}{2}}
\end{aligned}$$

Arithmetic means: which variant is best?

Is there any differences in accuracy between

1. two factors of arithmetic means
2. the arithmetic mean of a product

More precisely,

$$\begin{aligned}
[PQ]^{n+\frac{1}{2}} = P^{n+\frac{1}{2}}Q^{n+\frac{1}{2}} &\approx \frac{1}{2}(P^n + P^{n+1})\frac{1}{2}(Q^n + Q^{n+1})\\
[PQ]^{n+\frac{1}{2}} &\approx \frac{1}{2}(P^nQ^n + P^{n+1}Q^{n+1})
\end{aligned}$$

It can be shown (by Taylor series around $t_{n+\frac{1}{2}}$) that both approximations are $\mathcal{O}(\Delta t^2)$

Solution of nonlinear equations in the Crank-Nicolson scheme

No big difference from the Backward Euler case, just more terms:

- Identify the $F(u) = 0$ for the unknown u^{n+1}
- Apply Picard iteration or Newton's method to the PDE
- Identify the sequence of linearized PDEs and iterate

Discretization of 1D stationary nonlinear differential equations

Differential equation:

$$-(\alpha(u)u')' + au = f(u), \quad x \in (0, L)$$

Boundary conditions:

$$\alpha(u(0))u'(0) = C, \quad u(L) = D$$

Relevance of this stationary 1D problem

1. As stationary limit of a diffusion PDE

$$u_t = (\alpha(u)u_x)_x + au + f(u)$$

$(u_t \to 0)$

2. The time-discrete problem at each time level arising from a Backward Euler scheme for a diffusion PDE

$$u_t = (\alpha(u)u_x)_x + f(u)$$

(au comes from u_t, $a \sim 1/\Delta t$, $f(u) := f(u) - u^{n-1}/\Delta t$)

Finite difference discretizations

The nonlinear term $(\alpha(u)u')'$ behaves just as a variable coefficient term $(\alpha(x)u')'$ wrt discretization:

$$[-D_x\alpha D_x u + au = f]_i$$

Written out at internal points:

$$-\frac{1}{\Delta x^2}\left(\alpha_{i+\frac{1}{2}}(u_{i+1} - u_i) - \alpha_{i-\frac{1}{2}}(u_i - u_{i-1})\right) + au_i = f(u_i)$$

$\alpha_{i+\frac{1}{2}}$: two choices

$$\alpha_{i+\frac{1}{2}} \approx \alpha(\frac{1}{2}(u_i + u_{i+1}) = [\alpha(\overline{u}^x)]^{i+\frac{1}{2}}$$
$$\alpha_{i+\frac{1}{2}} \approx \frac{1}{2}(\alpha(u_i) + \alpha(u_{i+1})) = [\overline{\alpha(u)}^x]^{i+\frac{1}{2}}$$

The latter results in

$$[-D_x\overline{\alpha}^x D_x u + au = f]_i\,.$$

$$
-\frac{1}{2\Delta x^2}\left((\alpha(u_i)+\alpha(u_{i+1}))(u_{i+1}-u_i) - (\alpha(u_{i-1})+\alpha(u_i))(u_i-u_{i-1})\right)
$$
$$
+ au_i = f(u_i)
$$

Boundary conditions

- At $i = N_x$: $u_i = 0$.
- At $i = 0$: $\alpha(u)u' = C$

$$
[\alpha(u)D_{2x}u = C]_0
$$

$$
\alpha(u_0)\frac{u_1 - u_{-1}}{2\Delta x} = C
$$

The fictitious value u_{-1} can, as usual, be eliminated with the aid of the scheme at $i = 0$

The structure of the equation system

Structure of nonlinear algebraic equations:

$$
A(u)u = b(u)
$$

$$
\begin{aligned}
A_{i,i} &= \frac{1}{2\Delta x^2}(-\alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + a \\
A_{i,i-1} &= -\frac{1}{2\Delta x^2}(\alpha(u_{i-1}) + \alpha(u_i)) \\
A_{i,i+1} &= -\frac{1}{2\Delta x^2}(\alpha(u_i) + \alpha(u_{i+1})) \\
b_i &= f(u_i)
\end{aligned}
$$

Note:

- $A(u)$ is tridiagonal: $A_{i,j} = 0$ for $j > 1 + 1$ and $j < i - 1$.
- The $i = 0$ and $i = N_x$ equation must incorporate boundary conditions

The equation for the Neumann boundary condition

$i = 0$: insert

$$
u_{-1} = u_1 - \frac{2\Delta x}{\alpha(u_0)}
$$

in $A_{0,0}$. The expression for $A_{i,i+1}$ applies for $i = 0$, and $A_{i,i-1}$ for $i = 0$ does not enter the system.

The equation for the Dirichlet boundary condition

1. For $i = N_x$ we can use the Dirichlet condition as a separate equation

$$u_i = D, \quad i = N_x$$

2. Alternative: for $i = N_x$ we can substitute u_{N_x} in $A_{i,i}$ by D and have $N_x - 1$ equations.

Picard iteration

Use the most recently computed vaue u^- of u in $A(u)$ and $b(u)$:

$$A(u^-)u = b(u^-)$$

Tridiagonal system: use tridiagonal Gaussian elimination

Details: without Dirichlet condition equation

$N_x = 2$ and Dirichlet condition not as a separate equation:

$$\left(\begin{array}{cc} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{array}\right)\left(\begin{array}{c} u_0 \\ u_1 \end{array}\right) = \left(\begin{array}{c} b_0 \\ b_1 \end{array}\right)$$

$$\begin{aligned}
A_{0,0} &= \frac{1}{2\Delta x^2}(-\alpha(u_1^-) + 2\alpha(u_0^-) - \alpha(u_1^-)) + a \\
A_{0,1} &= -\frac{1}{2\Delta x^2}(\alpha(u_0^-) + \alpha(u_1^-)) \\
A_{1,0} &= -\frac{1}{2\Delta x^2}(\alpha(u_0^-) + \alpha(u_1^-)) \\
A_{1,1} &= \frac{1}{2\Delta x^2}(-\alpha(u_0^-) + 2\alpha(u_1^-) - \alpha(u_2)) + a \\
b_0 &= f(u_0^-) \\
b_1 &= f(u_1^-)
\end{aligned}$$

Note: substitute u_{-1} by Neumann condition cormula, subst. u_2 by D

Details: with Dirichlet condition equation

$N_x = 2$ and including the Dirichlet condition as a separate equation:

$$\left(\begin{array}{ccc} A_{0,0} & A_{0,1} & A_{0,2} \\ A_{1,0} & A_{1,1} & A_{1,2} \\ A_{2,0} & A_{2,1} & A_{2,2} \end{array}\right)\left(\begin{array}{c} u_0 \\ u_1 \\ u_2 \end{array}\right) = \left(\begin{array}{c} b_0 \\ b_1 \\ b_2 \end{array}\right)$$

with $A_{i,j}$ and b_i as before for $i, j = 1, 2$, keeping u_2 as unknown in $A_{1,1}$, and

$$
\begin{aligned}
A_{0,2} &= A_{2,0} = A_{2,1} = 0 \\
A_{1,2} &= -\frac{1}{2\Delta x^2}(\alpha(u_1) + \alpha(u_2)) \\
A_{2,2} &= 1,\ \ b_2 = D
\end{aligned}
$$

Newton's method; Jacobian (1)

Nonlinear eq.no i has the structure

$$
F_i = A_{i,i-1}(u_{i-1}, u_i)u_{i-1} + A_{i,i}(u_{i-1}, u_i, u_{i+1})u_i + \\
A_{i,i+1}(u_i, u_{i+1})u_{i+1} - b_i(u_i)
$$

Need Jacobian, i.e., need to differentiate $F(u) = A(u)u - b(u)$ wrt u. Example:

$$
\begin{aligned}
\frac{\partial}{\partial u_i}(A_{i,i}(u_{i-1}, u_i, u_{i+1})u_i) &= \frac{\partial A_{i,i}}{\partial u_i}u_i + A_{i,i}\frac{\partial u_i}{\partial u_i} \\
&= \frac{\partial}{\partial u_i}(\frac{1}{2\Delta x^2}(-\alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + a)u_i + \\
&\quad \frac{1}{2\Delta x^2}(-\alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + a) \\
&= \frac{1}{2\Delta x^2}(2\alpha'(u_i)u_i - \alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + a
\end{aligned}
$$

Newton's method; Jacobian (2)

The complete Jacobian becomes (make sure you get this!)

$$
\begin{aligned}
J_{i,i} &= \frac{\partial F_i}{\partial u_i} = \frac{\partial A_{i,i-1}}{\partial u_i}u_{i-1} + \frac{\partial A_{i,i}}{\partial u_i}u_i + A_{i,i} + \frac{\partial A_{i,i+1}}{\partial u_i}u_{i+1} - \frac{\partial b_i}{\partial u_i} \\
&= \frac{1}{2\Delta x^2}(-\alpha'(u_i)u_{i-1} + 2\alpha'(u_i)u_i - \alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + \\
&\quad a - \frac{1}{2\Delta x^2}\alpha'(u_i)u_{i+1} - b'(u_i) \\
J_{i,i-1} &= \frac{\partial F_i}{\partial u_{i-1}} = \frac{\partial A_{i,i-1}}{\partial u_{i-1}}u_{i-1} + A_{i-1,i} + \frac{\partial A_{i,i}}{\partial u_{i-1}}u_i - \frac{\partial b_i}{\partial u_{i-1}} \\
&= \frac{1}{2\Delta x^2}(-\alpha'(u_{i-1})u_{i-1} - (\alpha(u_{i-1}) + \alpha(u_i)) + \alpha'(u_{i-1})u_i) \\
J_{i,i+1} &= \frac{\partial A_{i,i+1}}{\partial u_{i-1}}u_{i+1} + A_{i+1,i} + \frac{\partial A_{i,i}}{\partial u_{i+1}}u_i - \frac{\partial b_i}{\partial u_{i+1}} \\
&= \frac{1}{2\Delta x^2}(-\alpha'(u_{i+1})u_{i+1} - (\alpha(u_i) + \alpha(u_{i+1})) + \alpha'(u_{i+1})u_i)
\end{aligned}
$$

Newton's method; nonlinear equations written out

$$F_i = -\frac{1}{2\Delta x^2}\left((\alpha(u_i)+\alpha(u_{i+1}))(u_{i+1}-u_i) - (\alpha(u_{i-1})+\alpha(u_i))(u_i-u_{i-1})\right) + au_i - f(u_i) = 0$$

At $i=0$, replace u_{-1} by formula from Neumann condition.

1. Exclude Dirichlet condition as separate equation: replace u_i, $i=N_x$, by D in F_i, $i=N_x-1$
2. Include Dirichlet condition as separate equation:

$$F_{N_x}(u_0,\ldots,u_{N_x}) = u_{N_x} - D = 0\,.$$

Note: The size of the Jacobian depends on 1 or 2.

Galerkin-type discretizations

- V: function space with basis functions $\psi_i(x)$, $i\in\mathcal{I}_s$
- Dirichlet conditionat $x=L$: $\psi_i(L)=0$, $i\in\mathcal{I}_s$ ($v(L)=0\ \forall v\in V$)
- $u = D + \sum_{j\in\mathcal{I}_s} c_j\psi_j$

Galerkin's method for $-(\alpha(u)u')' + au = f(u)$:

$$\int_0^L \alpha(u)u'v'\,\mathrm{d}x + \int_0^L auv\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x + [\alpha(u)u'v]_0^L,\quad \forall v\in V$$

Insert Neumann condition:

$$[\alpha(u)u'v]_0^L = \alpha(u(L))u'(L)v(L) - \alpha(u(0))u'(0)v(0) = -Cv(0)$$

The nonlinear algebraic equations

Find $u\in V$ such that

$$\int_0^L \alpha(u)u'v'\,\mathrm{d}x + \int_0^L auv\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x - Cv(0),\quad \forall v\in V$$

$\forall v\in V\ \Rightarrow\ \forall i\in\mathcal{I}_s$, $v=\psi_i$. Inserting $u = D+\sum_j c_j\psi_j$ and sorting terms:

$$\sum_j \left(\int\limits_0^L \alpha(D+\sum_k c_k\psi_k)\psi_j'\psi_i'\,\mathrm{d}x\right) c_j = \int\limits_0^L f(D+\sum_k c_k\psi_k)\psi_i\,\mathrm{d}x - C\psi_i(0)$$

This is a *nonlinear algebraic system*

Fundamental integration problem: how to deal with $\int f(\sum_k c_k\psi_k)\psi_i dx$ for unknown c_k?

- We do not know c_k in $\int_0^L f(\sum_k c_k\psi_k)\psi_i dx$ and $\int_0^L \alpha(\sum_k c_k\psi_k)\psi_i'\psi_j'\,\mathrm{d}x$
- Solution: numerical integration with approximations to c_k, as in $\int_0^L f(u^-)\psi_i dx$

> Next: want to do *symbolic integration* of such terms to see the structure of nonlinear finite element equations (to compare with finite differences)

Choose finite element basis functions

$$\psi_i = \varphi_{\nu(i)}, \quad i\in\mathcal{I}_s$$

Degree of freedom number $\nu(i)$ in the mesh corresponds to unknown number i (c_i). Model problem: $\nu(i)=i$, $\mathcal{I}_s=\{0,\ldots,N_n-2\}$

$$u = D + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)}$$

or with φ_i in the boundary function:

$$u = D\varphi_0 + \sum_{j\in\mathcal{I}_s} c_j\varphi_{j+1}$$

The group finite element method

Since u is represented by $\sum_j \varphi_j u(x_j)$, we may use the same approximation for $f(u)$:

$$f(u)\approx \sum_j f(x_j)\varphi_j$$

$f(x_j)$: value of f at node j. With u_j as $u(x_j)$, we can write

$$f(u)\approx \sum_j f(u_j)\varphi_j$$

This approximation is known as the *group finite element method* or the *product approximation* technique. The index j runs over all node numbers in the mesh.

What is the point with the group finite element method?

1. Complicated nonlinear expressions can be simplified to increase the efficiency of numerical computations.
2. One can derive *symbolic forms* of the difference equations arising from the finite element method in nonlinear problems. The symbolic form is useful for comparing finite element and finite difference equations of nonlinear differential equation problems.

Simplified problem for symbolic calculations

Simple nonlinear problem: $-u'' = u^2$, $u'(0) = 1$, $u'(L) = 0$.

$$\int_0^L u'v'\,\mathrm{d}x = \int_0^L u^2 v\,\mathrm{d}x - v(0),\quad \forall v \in V$$

Now,

- Focus on $\int u^2 v\,\mathrm{d}x$
- Set $c_j = u(x_j) = u_j$
  (to mimic finite difference interpretation of u_j)
- That is, $u = \sum_j u_j\varphi_j$

Integrating very simple nonlinear functions results in complicated expressions in the finite element method

Consider $\int u^2 v\,\mathrm{d}x$ with $u = \sum_k u_k\varphi_k$ and $v = \varphi_i$:

$$\int_0^L (\sum_k u_k\varphi_k)^2\varphi_i\,\mathrm{d}x$$

Tedious exact evaluation on uniform P1 elements:

$$\frac{h}{12}(u_{i-1}^2 + 2u_i(u_{i-1} + u_{i+1}) + 6u_i^2 + u_{i+1}^2)$$

Finite difference counterpart: u_i^2 (!)

Application of the group finite element method

$$\int_0^L f(u)\varphi_i\,\mathrm{d}x \approx \int_0^L (\sum_j \varphi_j f(u_j))\varphi_i\,\mathrm{d}x = \sum_j (\underbrace{\int_0^L \varphi_i\varphi_j\,\mathrm{d}x}_{\text{mass matrix } M_{i,j}})f(u_j)$$

Corresponding part of difference equation for P1 elements:

$$\frac{h}{6}(f(u_{i-1}) + 4f(u_i) + f(u_{i+1}))$$

Rewrite as "finite difference form plus something":

$$\frac{h}{6}(f(u_{i-1}) + 4f(u_i) + f(u_{i+1})) = h[f(u) - \frac{h^2}{6}D_xD_xf(u)]_i$$

This is like the finite difference discretization of $-u'' = f(u) - \frac{h^2}{6}f''(u)$

Lumping the mass matrix gives finite difference form

Lumped mass matrix (integrate at the nodes): M becomes diagonal and the finite element and difference method's treatment of $f(u)$ becomes identical!

Alternative: evaluation of finite element terms at nodes gives great simplifications

Idea: integrate $\int f(u)v\,\mathrm{d}x$ numerically with a rule that samples $f(u)v$ at the nodes only. This involves great simplifications, since

$$\sum_k u_k\varphi_k(x_\ell) = u_\ell$$

and

$$f(\sum_k u_k \underbrace{\varphi_k(x_\ell)}_{\delta_{k\ell}}) \underbrace{\varphi_i(x_\ell)}_{\delta_{i\ell}} = f(u_\ell)\delta_{i\ell}$$

($\delta_{ij} = 0$ if $i \neq j$ and $\delta_{ij} = 1$ if $i = j$)

Numerical integration of nonlinear terms

Trapezoidal rule with the nodes only gives the finite difference form of $[f(u)]_i$:

$$\int_0^L f(\sum_k u_k\varphi_k)(x)\varphi_i(x)\,\mathrm{d}x \approx h\sum_{\ell=0}^{N_n-1} f(u_\ell)\delta_{i\ell} - \mathcal{C} = hf(u_i)$$

($\mathcal{C}$: boundary adjustment of rule, $i = 0, N_n - 1$)

Finite elements for a variable coefficient Laplace term

Consider the term $(\alpha u')'$, with the group finite element method: $\alpha(u) \approx \sum_k \alpha(u_k)\varphi_k$, and the variational counterpart

$$\int_0^L \alpha(\sum_k c_k\varphi_k)\varphi_i'\varphi_j'\,\mathrm{d}x \approx \sum_k(\int_0^L \varphi_k\varphi_i'\varphi_j'\,\mathrm{d}x)\alpha(u_k) = \ldots$$

Further calculations (see text) lead to

$$-\frac{1}{h}(\frac{1}{2}(\alpha(u_i)+\alpha(u_{i+1}))(u_{i+1}-u_i)-\frac{1}{2}(\alpha(u_{i-1})+\alpha(u_i))(u_i-u_{i-1}))$$

= std finite difference discretization of $-(\alpha(u)u')'$ with an arithmetic mean of $\alpha(u)$

Numerical integration at the nodes

Instead of the group finite element method and exact integration, use Trapezoidal rule in the nodes for

$$\int_0^L \alpha(\sum_k u_k\varphi_k)\varphi_i'\varphi_j'\,\mathrm{d}x$$

Work at the cell level (most convenient with discontinuous φ_i'):

$$\int_{-1}^{1}\alpha(\sum_t \tilde{u}_t\tilde{\varphi}_t)\tilde{\varphi}_r'\tilde{\varphi}_s'\frac{h}{2}dX = \int_{-1}^{1}\alpha(\sum_{t=0}^{1}\tilde{u}_t\tilde{\varphi}_t)\frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\frac{2}{h}\frac{d\tilde{\varphi}_s}{dX}\frac{h}{2}dX$$
$$= \frac{1}{2h}(-1)^r(-1)^s\int_{-1}^{1}\alpha(\sum_{t=0}^{1}u_t\tilde{\varphi}_t(X))dX$$
$$\approx \frac{1}{2h}(-1)^r(-1)^s\alpha(\sum_{t=0}^{1}\tilde{\varphi}_t(-1)\tilde{u}_t)+\alpha(\sum_{t=0}^{1}\tilde{\varphi}_t(1)\tilde{u}_t)$$
$$= \frac{1}{2h}(-1)^r(-1)^s(\alpha(\tilde{u}_0)+\alpha(\tilde{u}^{(1)}))$$

Summary of finite element vs finite difference nonlinear algebraic equations

$$-(\alpha(u)u')'+au=f(u)$$

Uniform P1 finite elements:

- Group finite element or Trapezoidal integration at nodes: $-(\alpha(u)u')'$ becomes $-h[D_x\overline{\alpha(u)}^x D_x u]_i$
- $f(u)$ becomes $hf(u_i)$ with Trapezoidal integration or the "mass matrix" representation $h[f(u)-\frac{h}{6}D_xD_xf(u)]_i$ if group finite elements
- au leads to the "mass matrix" form $ah[u-\frac{h}{6}D_xD_xu]_i$

Real computations utilize accurate numerical integration

- Previous group finite element or Trapezoidal integration examples had one aim: derive symbolic expressions for finite element equations
- Real world computations apply numerical integration
- How to define Picard iteration and Newton's method from a variational form with numerical integration in real world computations?

Picard iteration defined from the variational form

$$-(\alpha(u)u')' + au = f(u), \quad x \in (0, L), \quad \alpha(u(0))u'(0) = C,\ u(L) = D$$

Variational form ($v = \psi_i$):

$$F_i = \int_0^L \alpha(u)u'\psi_i' \,\mathrm{d}x + \int_0^L au\psi_i \,\mathrm{d}x - \int_0^L f(u)\psi_i \,\mathrm{d}x + C\psi_i(0) = 0$$

Picard iteration: use "old value" u^- in $\alpha(u)$ and $f(u)$ and integrate numerically:

$$F_i = \int_0^L (\alpha(u^-)u'\psi_i' + au\psi_i) \,\mathrm{d}x - \int_0^L f(u^-)\psi_i \,\mathrm{d}x + C\psi_i(0)$$

The linear system in Picard iteration

$$F_i = \int_0^L (\alpha(u^-)u'\psi_i' + au\psi_i) \,\mathrm{d}x - \int_0^L f(u^-)\psi_i \,\mathrm{d}x + C\psi_i(0)$$

This is a linear problem $a(u, v) = L(v)$ with bilinear and linear forms

$$a(u, v) = \int_0^L (\alpha(u^-)u'v' + auv) \,\mathrm{d}x, \quad L(v) = \int_0^L f(u^-)v \,\mathrm{d}x - Cv(0)$$

The linear system now is computed the standard way.

The equations in Newton's method

$$F_i = \int_0^L (\alpha(u)u'\psi_i' + au\psi_i - f(u)\psi_i) \,\mathrm{d}x + C\psi_i(0) = 0, \quad i \in \mathcal{I}_s$$

Easy to evaluate right-hand side $-F_i(u^-)$ by numerical integration:

$$F_i = \int_0^L (\alpha(u^-)u'\psi_i' + au\psi_i - f(u^-)\psi_i) \,\mathrm{d}x + C\psi_i(0) = 0$$

(just known functions)

Useful formulas for computing the Jacobian

$$\frac{\partial u}{\partial c_j} = \frac{\partial}{\partial c_j}\sum_k c_k\psi_k = \psi_j$$

$$\frac{\partial u'}{\partial c_j} = \frac{\partial}{\partial c_j}\sum_k c_k\psi_k' = \psi_j'$$

Computing the Jacobian

$$\begin{aligned}
J_{i,j} = \frac{\partial F_i}{\partial c_j} &= \int_0^L \frac{\partial}{\partial c_j}(\alpha(u)u'\psi_i' + au\psi_i - f(u)\psi_i)\,\mathrm{d}x\\
&= \int_0^L ((\alpha'(u)\frac{\partial u}{\partial c_j}u' + \alpha(u)\frac{\partial u'}{\partial c_j})\psi_i' + a\frac{\partial u}{\partial c_j}\psi_i - f'(u)\frac{\partial u}{\partial c_j}\psi_i)\,\mathrm{d}x\\
&= \int_0^L ((\alpha'(u)\psi_j u' + \alpha(u)\psi_j')\psi_i' + a\psi_j\psi_i - f'(u)\psi_j\psi_i)\,\mathrm{d}x\\
&= \int_0^L (\alpha'(u)u'\psi_i'\psi_j + \alpha(u)\psi_i'\psi_j' + (a - f(u))\psi_i\psi_j)\,\mathrm{d}x
\end{aligned}$$

Use $\alpha'(u^-)$, $\alpha(u^-)$, $f'(u^-)$, $f(u^-)$ and integrate expressions numerically (only known functions)

Computations in a reference cell $[-1,1]$

$$\tilde{F}_r^{(e)} = \int_{-1}^{1}\left(\alpha(\tilde{u}^-)\tilde{u}^{-\prime}\tilde{\varphi}_r' + (a - f(\tilde{u}^-))\tilde{\varphi}_r\right)\det J\,\mathrm{d}X - C\tilde{\varphi}_r(0)$$

$$\tilde{J}_{r,s}^{(e)} = \int_{-1}^{1}(\alpha'(\tilde{u}^-)\tilde{u}^{-\prime}\tilde{\varphi}_r'\tilde{\varphi}_s + \alpha(\tilde{u}^-)\tilde{\varphi}_r'\tilde{\varphi}_s' + (a - f(\tilde{u}^-))\tilde{\varphi}_r\tilde{\varphi}_s)\det J\,\mathrm{d}X$$

$r, s \in I_d$ (local degrees of freedom)

How to handle Dirichlet conditions in Newton's method

- Newton's method solves $J(u^-)\delta u = -F(u^-)$
- δu is a *correction* to u^-
- If $u(x_i)$ has Dirchlet condition D, set $u_i^- = D$ in prior to the first iteration
- Set $\delta u_i = 0$ (no change for Dirichlet conditions)